Aix*Marseille université
Initiative d'excellence
Année 2019-20

Domaine Sciences et Technologies
Portail René Descartes

Programmation 1 : TP 9
Code UE : S14IN2I1

Projet Astéroïde : Partie 3

Ce TP poursuit le précédent. Il faut avoir terminer le TP8 avant de commencer ce sujet.

1 Classe projectile minimale

On souhaite ajouter au vaisseau la possibilité de tirer des projectiles capables de détruire les astéroïdes. Les projectiles seront des objets de la classe `Projectile`. Ils se déplacent en ligne droite à vitesse constante. Lorsqu'ils entrent en collision avec un astéroïde, le projectile et l'astéroïde disparaissent. Un projectile lancé depuis plus de 10 secondes devient inoffensif (il disparait).

1. Créer dans le package `game` une nouvelle classe `Projectile`.
2. Un projectile est caractérisé par :
   — sa position,
   — sa vélocité,
   — sa durée de vie restante.

   Ajouter les trois propriétés correspondantes dans la classe.
3. Ajouter un constructeur, initialisant la durée de vie à 10 secondes, et les autres propriétés selon les paramètres reçus.
4. La position devra être consultée par le reste du jeu ainsi que par les méthodes d'affichage.

   Ajouter donc un accesseur `getPosition`.
5. Ajouter une méthode `public void update`, qui met à jour la durée de vie et la position du projectile.
   Notez que le projectile ne subit pas d'accélération, sa vélocité est donc constante.
6. Ajouter un prédicat `isAlive` permettant de vérifier si la durée de vie du projectile n'est pas écoulée.
7. Dans la classe `Spaceship`, ajouter une méthode `public Projectile fire()`.
   Le nouveau projectile doit être placé à une distance de 30 (pixels) en avant du vaisseau. Il se déplace à 100 pixels par seconde droit devant le vaisseau. Puisque le vaisseau est possiblement en mouvement, il hérite aussi de la vélocité du vaisseau. Sa vélocité initiale doit donc être la somme de la vélocité du vaisseau d'une part, et d'un vecteur de même direction que le vaisseau et de norme 100 d'autre part.

2 Ajout des projectiles au modèle

Le vaisseau peut tirer de multiples projectiles. Il faut enregistrer leur position dans l'état du jeu, qui est représenté par la classe `Space`.

1. Dans la classe `Space`, ajouter une propriété `projectiles` pour contenir les projectiles actifs.
2. S'assurer que `projectiles` soit correctement initialisé.
   Au début, l'espace ne contient aucun projectile.
3. Dans le méthode `update` de `Space`, faire en sorte que la position de chaque projectile soit mise à jour.
   Pour cela, créer une méthode `updateProjectiles(double dt)`, qui sera appelé dans `update`, et dont le rôle est de mettre à jour la position et la durée de vie de chaque projectile.
4. Ajouter une méthode publique `getProjectiles` permettant d'obtenir la liste des projectiles actifs.
5. Ajouter une méthode `public void addProjectile`, prenant un projectile en paramètre, et ajoutant ce projectile à la liste des projectiles.

3 Afficher les projectiles

Nous avons modifié le modèle pour qu'il contienne des projectiles. Il nous faut maintenant les afficher à l'écran.

1. Dans la classe `viewModel`, ajouter une méthode `public List<Projectile> getProjectiles()` retournant la liste des projectiles du modèle.
2. Dans la même classe, ajouter une méthode `public void fireSpaceshipGun()`, qui fait tirer un projectile au vaisseau et ajoute le projectile obtenu à l'espace.
3. On se place dorénavant dans la classe `CanvasView`.

   Ajouter une méthode `private void render(Projectile bullet)` prenant un projectile en argument. Cette méthode doit afficher le projectile dans la zone de dessin.

   Pour rappel, la classe `CanvasView` contient une propriété `GraphicsContext context` qui représente la zone de dessin et les opérations de dessin que nous pouvons faire. Il nous faut :

   (a) une première instruction pour changer la couleur du pinceau (de la même façon que dans la méthode `renderAsteroid`). Choisissez la couleur qui vous plait.

   (b) Une deuxième instruction pour dessiner un disque de diamètre 4 : `fillOval`. Consultez la documentation (`Ctrl+Q` ou bien sûr Internet [1]), pour connaître la signification des paramètres de cette méthode. Attention à bien positionner le centre du projectile.
4. Modifier la méthode `render` pour qu'elle appelle la méthode `render` pour tous les projectiles.
5. Finalement, aller dans la classe `View`. Modifier la méthode `handleKeyPressed` pour que lorsque la touche `SPACE` est appuyée, le vaisseau lance un projectile.
6. Démarrer le programme. Vérifier que tout fonctionne correctement, ou faites les ajustements nécessaires.

4 Gérer la fin de vie des projectiles

Pour l'instant les projectiles restent actifs indéfiniment. On veut les faire disparaître dès que leur temps de vie est écoulé. Les modifications suivantes ont toutes lieu dans la classe `Space`.

1. Ajouter une méthode `private List<Projectile> getDeadProjectiles()`.
   Dans cette méthode :
   (a) créer une liste vide,
   (b) pour chaque projectile, s'il est mort, l'ajouter à la liste,
   (c) puis retourner la liste des projectiles morts.
2. Ajouter une méthode `private void removeDeadProjectiles()`.
   Dans cette méthode, pour tout projectile mort, retirez le projectile mort de la liste des projectiles actifs, en utilisant la méthode de liste adéquate.
3. Modifier la méthode `update` : à la fin de cette méthode, retirer les projectiles morts de la liste des projectiles actifs à l'aide de la méthode écrite juste avant.
4. Relancer le programme et vérifier le bon fonctionnement de cette modification.

1. `https://docs.oracle.com/javase/12/javafx/api/javafx/scene/canvas/GraphicsContext.html`

5 Destructions des astéroïdes

Nous souhaitons faire en sorte que l'impact d'un projectile sur un astéroïde fragmente cet astéroïde en plusieurs astéroïdes plus petits. Si un projectile touché est déjà tout petit, alors il sera simplement détruit. Pour commencer, faisons en sorte que l'impact d'un projectile détruise l'astéroïde (sans création de fragments).

1. Dans la classe `Projectile`, ajouter un prédicat prenant un astéroïde en paramètre, et déterminant si le projectile est contenu dans la zone de cet astéroïde.
2. Que doit faire le modèle à chaque mise-à-jour, concernant les projectiles ?
   (a) Déplacer les projectiles selon le pas de temps `dt` et mettre à jour leur durée de vie,
   (b) chercher tous les projectiles et tous les astéroïdes en contact,
   (c) supprimer tout projectile en contact avec un astéroïde,
   (d) remplacer tous les astéroïdes en contact avec un projectile par des astéroïdes plus petits.
   (e) supprimer tous les projectiles en fin de vie.

   Créer une méthode `processProjectiles(double dt)` dans `Space`. Réaliser dans cette méthode les points 1 et 5. Modifier la méthode `update` pour quelle utilise `processProjectile`.

   On va rajouter les points 2, 3 et 4 dans les questions suivantes.
3. Créer deux variables contenant des ensembles vides dans la méthode `processProjectiles`.
   Le premier contiendra les projectiles en contact avec un astéroïde, le second contiendra les astéroïdes touchés par un projectile.
4. Toujours dans la méthode `processProjectiles`, à la suite des instructions, ajouter des instructions testant pour chaque projectile et chaque astéroïde, si le projectile est contenu dans l'astéroïde. Dans ce cas, on ajoute le projectile et l'astéroide aux ensembles de projectiles et d'astéroïdes en contact.
5. À la suite, ajouter des instructions retirant chaque projectile touchant un astéroïde (les projectiles de l'ensemble construit juste avant). Vous pouvez utiliser une boucle `for` sur un ensemble.
6. Encore à la suite, ajouter de la même façon des instructions retirant chaque astéroïde touché par au moins un projectile.
   On s'occupera de fragmenter les astéroïdes touchés plus tard, pour l'instant on se contente de les faire disparaître.
7. Vous obtenez ainsi une méthode `processProjectiles` complète mais longue et complexe. Nous souhaitons modifier notre nouvelle méthode `processProjectiles` pour la rendre plus lisible. Spécifiquement, nous voulons que ses instructions soient :

```
private void processProjectiles(double dt) {
  updateProjectiles(dt);
  Set<Bullet> hittingProjectiles = new HashSet<>();
  Set<Asteroid> hittedAsteroids = new HashSet<>();
  findProjectileHits(hittingProjectiles, hittedAsteroids);
  remove(hittingProjectiles);
  fragment(hittedAsteroids);
}
```

   On peut utiliser l'outil d'extraction de méthode (`Refactor > Extract > Method...`) pour prendre des instructions et ordonner à IntelliJ de les transformer en une méthode indépendante.

En utilisant l'extraction de méthode et le renommage (raccourci `Shift + F6`), refactoriser la méthode jusqu'à ce qu'elle soit telle que ci-dessus.

8. Vérifier que le programme fonctionne correctement, avec la nouvelle fonctionnalité.

6 Fragmentation des astéroïdes

On ajoute maintenant la fonctionnalité de fragmentation des astéroïdes : un astéroïde détruit est remplacé par des astéroïdes plus petits. En dessous d'une certaine taille, un astéroïde détruit disparaît sans se fragmenter (ce qui permet au jeu de pouvoir se terminer).

1. Dans la classe `Asteroid`, ajouter une méthode `fragments`.
   Pour l'instant, la méthode retourne une liste vide, nous la complèterons plus tard, pour qu'elle retourne une liste d'astéroïdes plus petits correspondant au fragment de l'astéroïde une fois détruit.
2. La propriété `size` permet de connaître la taille approximative de l'astéroïde. La classe `Space` contient déjà la définition de la taille initiale d'un astéroïde, sous la forme d'une constante.

   Ajouter d'autres constantes dans `Space` :

   — la taille en-dessous de laquelle un algorithme détruit ne génère pas de fragments,
   — le nombre de fragments lorsqu'un astéroïde de taille suffisante est détruit,
   — le ratio de la taille d'un fragment sur la taille de l'astéroïde détruit (qui doit être inférieur à 1).
3. Revenir à la méthode `fragments` de la classe `Asteroid`.

   Ajouter des instructions pour créer une liste et la remplir d'astéroïdes correspondant aux fragments créés par la destruction.

   Pour créer un fragment, on utilise le générateur aléatoire `Space.generator`. Les fragments créés doivent être dans la position de l'astéroïde dont ils proviennent, et avoir une taille égale à la taille de l'astéroïde dont ils proviennent fois le ratio de taille défini ci-dessus.
4. Dans la classe `Space`, trouver la méthode chargée de gérer la destruction de l'astéroïde. Ajouter à cette méthode les instructions pour ajouter les fragments de l'astéroïde détruit à l'espace.
5. Vérifier le fonctionnement du programme.
6. Dans la classe `Asteroid`, ajouter les instructions pour que lorsque l'astéroïde détruit est suffisamment petit, il ne se fragmente pas, mais disparaisse simplement sans trace.
7. Vérifier le fonctionnement du programme.

7 Score

On améliore maintenant la façon dont le score est calculé. On veut favoriser les tireurs les plus efficaces et réguliers, capables de détruire rapidement tous les astéroïdes. Voici le système que nous voulons implémenter.

Le score est augmenté à chaque fois qu'un astéroïde est détruit. Pour chaque astéroïde détruit, le joueur marque 10 points de base. Pour récompenser la régularité du joueur, ces 10 points sont multipliés par un entier strictement positif, le *multiplicateur*. Le multiplicateur vaut initialement 1. Il est augmenté de 1 chaque fois qu'un astéroïde est détruit (après avoir compté son score). Il redescend de 1 si pendant 3 secondes, aucun astéroïde n'est touché. Il ne peut pas descendre en-dessous de 1.

1. Créer dans le package `game` une nouvelle classe `Score`.

2. Lui ajouter une propriété pour le score (un `double`) initialement nul, et un accesseur.
3. Ajouter aussi une méthode `public void update(double dt)`. Pour l'instant cette méthode ne fait rien.
4. Dans la classe `Space`, modifier la propriété `score`, pour que son type devienne `Score`. Supprimer la méthode `updateScore`, et modifier la méthode `update` en remplaçant l'appel `updateScore(score)`.
5. Mettre à jour la méthodes `getScore` de `Space` et de `ViewModel`, pour que celle de `Space` retourne un objet de type `Score`. Celle de `viewModel` retourne toujours un `double`.
6. Vérifier que le programme fonctionne. Le score devrait être 0 et ne jamais changer.
7. Dans la classe `Score`, ajouter une méthode privée `addPoints(double points)`, permettant d'augmenter le score d'un nombre donné de points.
8. Dans la classe `Score`, ajouter une méthode public `notifyAsteroidHit`. Cette méthode utilise `addPoints` pour augmenter le score de 10 points.
9. Dans la classe `Space`, faire en sorte que lorsqu'un astéroïde est fragmenté ou détruit, le score augmente de 10 points.
10. Vérifier le bon fonctionnement du programme.
11. Ajouter une propriété `int multiplier` à la classe `Score`, initialement 1. Ajoutez un accesseur pour cette propriété.
12. Modifier la méthode `addPoints` de `Score` pour que le nombre de points ajoutés soit multiplié par le multiplicateur.
13. Créez une méthode `addMultiplier` pour modifier le multiplicateur.
    Elle prend en argument un entier déterminant de combien le multiplicateur change (négatif si le multiplicateur décroit). Attention, le multiplicateur doit rester strictement positif.
14. Modifier `Score` de sorte que le multiplicateur augmente de 1 lorsqu'un astéroïde est touché.
15. Dans la classe `ViewModel`, ajouter une méthode `getScoreMultiplier` retournant le multiplicateur du score.
16. Dans la classe `CanvasView`, consultez les instructions permettant d'afficher le score, dans la méthode `renderScore`.

    Ajouter des instructions pour que la valeur du multiplicateur s'affiche aussi, en dessous du score, précédée du symbole $\times$.

17. Vérifiez le bon comportement du programme.
18. On veut faire en sorte que le multiplicateur diminue de 1 au bout de 3 secondes consécutivement sans événement. Pour cela, on ajoute une sorte de compte à rebour.

    Ajoutez une propriété `multiplierTimer` permettant de mesurer le temps restant avant de devoir diminuer le multiplicateur.

19. Faire en sorte que lorsque le multiplicateur est modifié, le compte à rebour `multiplierTimer` est remis à 3.
20. Modifier `update` pour que `multiplierTimer` décroit lorsque le temps passe. Lorsqu'il atteint 0, diminuez de 1 le multiplicateur.

21. Vérifier le bon comportement du programme.

22. Vous assurer que les instructions des méthodes ne fassent jamais référence à des littéraux numériques.
Pour cela, pour chaque littéral numérique, créez une constante (`private final static double CONSTANT_NAME = ...;`). Vous pouvez utilisez le menu `Refactor > Extract > Constant...` en sélectionnant d'abord le littéral à transformer en constante, pour qu'IntelliJ crée la constante correspondante.